28/1/2016 Úvod do databáz, skúškový test, max 25 bodov, 90 min

1. Uvažujte databázu bez duplikátov a null hodnôt: capuje(Krcma, Alkohol), lubi(Pijan, Alkohol), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).
Platí: Idn → Pijan, Krcma; Idn, Alkohol → Mnozstvo; Mnozstvo > 0.

a) Nájdite dvojice [P, A] také, že pijan P ľúbi alkohol A; a zároveň alkohol A sa čapuje v každej krčme, v ktorej P vypil niektorý zo svojich obľúbených alkoholov. Sformulujte dotaz v Datalogu **(2)** a relačnom kalkule **(2)**.

Datalog:

```
answer(P, A) ←
        lubi(P, A),
        not vo_necapuje(P, A).

vo_necapuje(P, A) ←
        lubi(P, A),
        navstivil(I, P, K),
        vypil(I, A2, _),
        lubi(P, A2),
        not capuje(K, A).
```

Relačný kalkul:

```
{[P, A]:
        lubi(P, A) ∧
        ¬
        (∃K ∃I ∃A2 ∃M
                navstivil(I, P, K) ∧ vypil(I, A2, M) ∧ lubi(P, A2) ∧
                ¬
                capuje(K, A)
        )
}
```

b) Nájdite trojice [K, A, N] s N>0, kde N je počet návštev krčmy K, pri ktorých sa vypil len alkohol A. Sformulujte dotaz v Datalogu **(2)** a SQL **(2)**.

```
answer(K, A, N) ←
        subtotal(data(K, A, I), [K, A], [N = count(I)]).

data(K, A, I) ←
        navstivil(I, _, K),
        vypil(I, A, _),
        not vypity_iny(I, A).

vypity_iny(I, A) ←
        vypil(I, A, _),
        vypil(I, A2, _),
        not A = A2.
```

SQL:

```
select n.Krcma as K, v.Alkohol as A, count(n.Idn) as N       /* count(distinct n.Idn) tam môže, ale nemusí byť */
from navstivil n, vypil v
where n.Idn = v.Idn and not exists (
        select *
        from vypil v2
        where v.Idn = v2.Idn and v.Alkohol <> v2.Alkohol)
group by n.Krcma, v.Alkohol
```

2. a) Definujte pojem bezstratovej dekompozície relačnej schémy. **(1)**

Dekompozícia $r_1, ..., r_N$ relačnej schémy r je bezstratová (spája sa bezstratovo), keď pre každú populáciu relácie r platí $r_1 \bowtie r_2 \bowtie ... \bowtie r_N = r$. (Relácie v tomto kontexte chápeme ako množiny, duplikáty ignorujeme.)

b) Rozhodnite či platí tvrdenie (r, $r_1, ..., r_N$ chápte ako množiny atribútov):
„Ak $(r_1 \cup r_2 \cup ... \cup r_N = r) \wedge ((r_1 \cap r_2 \cap ... \cap r_N) \rightarrow r)$, tak $r_1, ..., r_N$ je bezstratová dekompozícia r.“ Dokážte alebo uveďte konkrétny kontrapríklad. **(2)**

Áno, platí.

To zjednotenie v prvom konjunkte hovorí len to, že $r_1, ..., r_N$ je nejaká dekompozícia r (každý atribút r je v niektorej r_i). Druhý konjunkt hovorí, že množina spoločných atribútov $r_1, ..., r_N$ je nadkľúčom v r; a tým pádom je tiež nadkľúčom v $r_1, ..., r_N$, keďže každá $r_1, ..., r_N$ je podmnožinou r.

Dokážeme, že platí $r_1 \bowtie r_2 \bowtie ... \bowtie r_N = r$. Nech **X** je množina spoločných atribútov relácií tej dekompozície, t.j. $\mathbf{X} = r_1 \cap r_2 \cap ... \cap r_N$. Pre i = 1, ..., N, nech $\mathbf{Y_i}$ označuje množinu všetkých atribútov r_i, ktoré nie sú v **X**. Nech $\mathbf{Y} = \mathbf{Y_1} \cup ..., \cup \mathbf{Y_N}$. Platí $\mathbf{X} \rightarrow \mathbf{Y}$ (**X** je nadkľúčom v r) a tiež platí $\mathbf{X} \rightarrow \mathbf{Y_i}$ (**X** je nadkľúčom v r_i).

Chceme dokázať $r_1(\mathbf{X}, \mathbf{Y_1}) \bowtie r_2(\mathbf{X}, \mathbf{Y_2}) \bowtie ... \bowtie r_N(\mathbf{X}, \mathbf{Y_N}) = r(\mathbf{X}, \mathbf{Y})$.

Uvažujme ľubovoľnú n-ticu t = [**x**, **y**], ktorá patrí do r (**x** sú hodnoty atribútov **X**, **y** sú hodnoty atribútov **Y**). Keďže $\mathbf{X} \rightarrow \mathbf{Y}$, do r nepatrí žiadna iná n-tica, ktorá sa s t zhoduje na hodnotách atribútov z **X**. Keďže platí $\mathbf{X} \rightarrow \mathbf{Y_i}$ (i=1, ..., N), v každej relácii r_i existuje práve 1 n-tica [**x**, $\mathbf{y_i}$], kde $\mathbf{y_i}$ je projekcia hodnôt **y** na príslušné atribúty r_i. Z definície prirodzeného joinu vyplýva, že t patrí do $r_1 \bowtie r_2 \bowtie ... \bowtie r_N$.

Uvažujme ľubovoľnú n-ticu t, ktorá patrí do $r_1(\mathbf{X}, \mathbf{Y_1}) \bowtie r_2(\mathbf{X}, \mathbf{Y_2}) \bowtie ... \bowtie r_N(\mathbf{X}, \mathbf{Y_N})$. Keďže $r_1, ..., r_N$ sa joinujú podľa **X**, t je tvaru [**x**, **y**], kde **x** sú hodnoty atribútov **X** a **y** sú hodnoty atribútov $\mathbf{Y_1} \cup ..., \cup \mathbf{Y_N}$. Keďže pre i = 1, ..., N, **X** je nadkľúčom v r_i, v každej r_i existuje práve jedna n-tica s hodnotami **x** na atribútoch **X**. Keďže $\mathbf{Y} = \mathbf{Y_1} \cup ..., \cup \mathbf{Y_N}$, n-tica t = [**x**, **y**] patrí aj do r(**X**, **Y**).

Dokázali sme, že ak nejaká n-tica patrí do toho joinu, tak patrí aj do r. Dokázali sme, že ak nejaká nejaká n-tica patrí do r, tak patrí aj do toho joinu. QED

c) Uvažujte relačnú schému r(A, B, C, D, E, F, G) s funkčnými závislosťami
ABCD → EF; ABE → FG; ABDG → CF; G → BD.
Rozhodnite či dekompozícia r1(A, B, C, D, E, G), r2(A, C, F, G) je v tretej normálnej forme. Zdôvodnite. **(2)**

Relačná schéma [r, F] je v tretej normálnej forme (3NF), ak F^+ neobsahuje funkčnú závislosť $X \rightarrow A$, kde A je jednoduchý atribút, $A \notin X$, X nie je nadkľúč r a A nepatrí do žiadneho kľúča r.
Dekompozícia je v 3NF, keď všetky relačné schémy tej dekompozície sú v 3NF.

Nájdime všetky kľúče r. Atribút A je v každom kľúči, lebo nie je na pravej strane žiadnej funkčnej závislosti. Jednoprvkové kľúče nie sú. Jediný dvojprvkový kľúč je AG. Ostatné kľúče nájdeme kompletným prehľadávaním. Všetky kľúče r sú: ABCD, ABE, AG.

Teda každý atribút okrem F patrí do niektorého kľúča r. Tým pádom r1 je v 3NF, lebo obsahuje iba kľúčové atribúty. V r2 môže 3NF porušiť len platná netriviálna funkčná závislosť s atribútom F na pravej strane a niektorými 2 atribútmi na ľavej strane. AG→F ani AG→C neporušujú 3NF, lebo AG je nadkľúč v r2. $\{A, C\}^* = \{A, C\}$. $\{C, G\}^* = \{C, G\}$. r2 je v 3NF.

Áno, daná dekompozícia je v tretej normálnej forme.

3. SQL príkaz *create assertion* definuje integritné obmedzenie. Systém dovolí aktualizáciu databázy len vtedy, keď sa podmienka v klauze *check* vyhodnotí po aktualizácii ako *true* (vyhodnocuje sa rovnako ako podmienka v klauze *where* príkazu *select*). Uvažujte reláciu r(A, B, C, D) bez duplikátov a *null* hodnôt.
a) Rozhodnite či nasledujúci príkaz vynúti splnenie funkčnej závislosti AB→C v relácii r (a iba to, t.j. okrem tohto nevynúti nič iné). Zdôvodnite. **(2)**
create assertion fd_ab_c check (
not exists (select A, B from r group by A, B having count(C) > 1));

Nie. Tento assertion síce vynúti splnenie funkčnej závislosti AB→C, ale vynúti aj niečo viac (vynúti splnenie funkčnej závislosti AB→CD).

Uvažujme reláciu r(A, B, C, D) = {[1, 1, 0, 0], [1, 1, 0, 1]}. Výsledkom toho *select* je {[1, 1]}, čo je neprázdna relácia ((lebo count(C) pre grupu [1, 1] je 2). Výsledkom toho *not exists* je teda FALSE, hoci funkčná závislosť AB→C v relácii r porušená nie je.

b) Napíšte bez grupovania a aritmetickej agregácie príkaz *assertion*, ktorý vynúti (iba) splnenie funkčnej závislosti AB→C (prvý riadok skopírujte z úlohy a)). **(2)**

Definícia funkčnej závislosti hovorí toto: AB→C platí v r(A, B, C, D), ak pre každú populáciu relácie r platí
¬ (∃A1 ∃A2 ∃B1 ∃B2 ∃C1 ∃C2 ∃D1 ∃D2
r(A1, B1, C1, D1) ∧ r(A2, B2, C2, D2) ∧ A1 = A2 ∧ B1 = B2 ∧ C1 ≠ C2
)

```
create assertion fd_ab_c check (
    not exists (
        select *
        from r r1, r r2
        where r1.A = r2.A and r1.B = r2.B and r1.C <> r2.C)
    )
)
```

c) Rozhodnite či vynechanie predpokladu absencie *null* hodnôt v relácii r ovplyvní riešenie úlohy b). Ak nie, vysvetlite prečo. Ak áno, vysvetlite rozdiel na vhodnom príklade a vyriešte úlohu b) pre reláciu r, ktorá môže obsahovať aj hodnoty *null*. **(2)**

Neovplyvní. Assertion z úlohy b) vynúti splnenie funkčnej závislosti AB→C bez ohľadu na prítomnosť null hodnôt. Vyplýva to z formálnej definície funkčnej závislosti a interpretácie null hodnôt v SQL.

V riešení úlohy b) je uvedená definícia funkčnej závislosti. Porušenie platnosti AB→C teda znamená splnenie formuly ∃A1 ∃A2 ∃B1 ∃B2 ∃C1 ∃C2 ∃D1 ∃D2
r(A1, B1, C1, D1) ∧ r(A2, B2, C2, D2) ∧ A1 = A2 ∧ B1 = B2 ∧ C1 ≠ C2.
Ak dosadíme null do A1 alebo A2 alebo B1 alebo B2, tak túto formulu nesplníme. Dôvodom je, že
null = <čokoľvek> je v SQL null (nie true). Taktiež null ≠ <čokoľvek> je v SQL null (nie true).
Taktiež null ∧ <čokoľvek> je v SQL null (nie true). Taktiež ¬ null je v SQL null (nie true).
Teda záznamy r obsahujúce null v ľubovoľnom z atribútov A, B, C funkčnú závislosť AB→C neporušia. Ani hodnota null atribútu D túto funkčnú závislosť neporuší.

Môžeme polemizovať o iných významoch null hodnôt. ***Null môžeme chápať napríklad ako neexistujúcu hodnotu****, t.j. po dosadení akýchkoľvek konštánt z domény za null hodnoty v n-tici nebude tá n-tica patriť do r (null reprezentuje napríklad rodné meno muža).* ***Aj pri takomto chápaní bude riešenie z úlohy b) korektné.***

Null hodnoty môžeme chápať aj tak, že nahrádzajú existujúce, ale neznáme hodnoty. *Inak povedané, každé null je možné nahradiť nejakou non-null hodnotou z domény tak, že daná n-tica bude patriť do r (null reprezentuje napríklad evidenčné číslo knihy, ktoré dosiaľ nebolo pridelené, hoci tú knihu už máme na sklade). Toto sa zdá byť dosť populárna interpretácia, má však svoje úskalia.* ***Riešenie z úlohy b) v tomto prípade fungovať nebude*** *(prinajmenšom nie vždy bude fungovať). Problematické sú nasledujúce inštancie r, kde napriek assertion z úlohy b) funkčná závislosť $AB \rightarrow C$ môže byť porušená:*

r = {[1, 1, null, 1], [1, 1, 1, 1]},
r = {[1, 1, null, 1], [1, 1, null, 0]},
r = {[1, 1, null, 1], [1, 1, null, null]},
r = {[null, 1, 1, 1], [1, 1, 0, 1]},
r = {[null, 1, 1, 1], [null, 1, 0, 1]},
r = {[1, null, 1, 1], [null, 1, 0, 1]},
...

Priamočiara možnosť ako testovať, či je nejaká hodnota null, je použitie podmienky „... is null“. Riešenie z úlohy b) sa síce dá modifikovať tak, aby postihovalo všetky tieto kontrapríklady—je to však dosť prácne, lebo treba rozlišovať medzi „... = null“ (čo vždy vráti null) a „... is null“ (čo vráti true keď prvý operand je null, inak vráti false). Aby sa toto trošku zjednodušilo, mnohé SQL systémy ponúkajú možnosť automatickej konverzie podmienok „... = null“ na „... is null“. Napríklad, v PostgreSQL na to slúži konfiguračná systémová funkcia transform_null_equals().

Použitie null hodnôt prináša problémy a oplatí sa zvážiť ako sa mu vyhnúť.

4. a) Popíšte čo najpresnejšie všeobecný algoritmus obnovy (bez checkpointov). **(2)**

Všeobecný algoritmus obnovy číta log-file najskôr zostupne (od konca log-file k začiatku). Počas tohto prechodu aktualizuje zoznamy redo_list a undo_list (undo_list aj redo_list sú na začiatku prázdne) a zároveň robí UNDO operácie pre transakcie z undo_list. Keď prečíta začiatok log-file (BOF), začne vzostupný prechod, pri ktorom vykonáva REDO operácie pre transakcie z redo_list. Keď prečíta koniec logfile (EOF), tak skončí a systém začne normálnu prevádzku.

Zoznamy sa pri zostupnom prechode aktualizujú takto: po prečítaní záznamu *write* od transakcie T, ktorá nie je v žiadnom zozname, sa T pridá do undo_list. Po prečítaní záznamu *commit* od transakcie T, ktorá nie je v žiadnom zozname, sa T pridá do redo_list. Po prečítaní start od transakcie T, ktorá je v undo_list, sa T odstráni z undo_list.
Zoznamy sa pri vzostupnom prechode aktualizujú takto: po prečítaní záznamu *commit* od transakcie T, ktorá je v redo_list, sa T odstráni z redo_list. Vzostupný prechod sa môže ukončiť hneď po vyprázdnení redo_list, t.j. ešte pred prečítaním EOF.

b) Databázový systém spadol. Pred jeho opätovným spustením obsahuje log-file nasledujúce záznamy: [T1, start], [T1, X, 1, 2], [T2, start], [T3, start], [T3, Y, 3, 2], [T3, commit], [T2, X, 2, 1], [T1, Y, 2, 0], [T2, commit]. Log-file rastie zľava doprava. Záznamy týkajúce sa operácií write obsahujú na treťom mieste starú hodnotu, na štvrtom mieste novú hodnotu. Okomentujte po každom zázname prečítanom z log-file akcie, ktoré systém urobí počas obnovy. **(2)**

Inicializácia:
undo_list = {};
redo_list = {};

Zostupný prechod:
[T2, commit]: redo_list = {T2};
[T1, Y, 2, 0]: undo_list = {T1}; write(Y, 2);
[T2, X, 2, 1]:
[T3, commit]: redo_list = {T2, T3};
[T3, Y, 3, 2]:
[T3, start]:
[T2, start]:
[T1, X, 1, 2]: write(X, 1);
[T1, start]: undo_list = {};
[BOF]

Vzostupný prechod:
[T1, start]:
[T1, X, 1, 2]:
[T2, start]:
[T3, start]: /* vzostupný prechod môže začať až tu, ak si pri zostupnom prechode zapamätáme aj záznam s najstarším REDO */
[T3, Y, 3, 2]: write(Y, 2);
[T3, commit]: redo_list = {T2};
[T2, X, 2, 1]: write(X, 1);
[T1, Y, 2, 0]:
[T2, commit]: redo_list = {};
[EOF]

c) Mohol log-file z úlohy b) vzniknúť pri bežnej prevádzke v systéme, ktorý používa na izoláciu transakcií striktné dvojfázové zamykanie? Vysvetlite. **(2)**

Nie. Striktné dvojfázové zamykanie generuje len rozvrhy, ktoré neobsahujú dirty reads ani dirty writes. Lenže scheduler v tomto systéme urobil dirty write, čo je vidieť zo záznamov [T1, X, 1, 2]→[T2, X, 2, 1]. V čase, keď transakcia T2 písala do X, nebola T1 commitovaná.